第41卷第5期
2022年10月

红外与毫米波学报
J. Infrared Millim. Waves

Vol. 41, No. 5
October, 2022

文章编号：1001-9014(2022)05-0914-09 DOI:10. 11972/j. issn. 1001-9014. 2022. 05. 016

An ultra-efficient streaming-based FPGA accelerator for infrared target detection

CHEN Shao-Yi[1,2,3,4], TANG Xin-Yi[2,3,4], WANG Jian[2,3,4], HUANG Jing-Si[1,2,3,4], LI Zheng[2,3,4*]

(1. School of Information Science and Technology, Shanghai Tech University, Shanghai 201210, China;
2. Shanghai Institute of Technical Physics, Chinese Academy of Sciences, Shanghai 20083, China;
3. University of Chinese Academy of Sciences, Beijing 100049, China;
4. Key Laboratory of Infrared System Detection and Imaging Technology, Chinese Academy of Sciences, Shanghai 200083, China)

Abstract: Object detection algorithm based on deep learning has achieved great success, significantly better than the effect of traditional algorithms, and even surpassed human in many scenarios. Unlike RGB cameras, infrared cameras can see objects even in the dark, which can be used in many fields like surveillance and autonomous driving. In this paper, a lightweight target detection algorithm for embedded devices is proposed, which is accelerated and deployed using Xilinx Ultrascale+MPSoC FPGA ZU3EG. The accelerator runs at a 350 MHz frequency clock with throughput of 551 FPS and power of only 8. 4 W. The intersection over union (IoU) of the algorithm achieves an accuracy of 73. 6% on FILR datasets. Comparing with the previous work, the accelerator design improves performance by 2. 59× and reduces 49. 02% of the power consumption.

Key words: infrared imaging, embedded software, real-time systems, Field Programmable Gate Array (FPGA), convolutional neural network

一种用于红外目标检测的高效流水线式FPGA加速器

陈少毅[1,2,3,4]，汤心溢[2,3,4]，王 健[2,3,4]，黄静思[1,2,3,4]，李 争[2,3,4*]

(1. 上海科技大学,上海 201210;
2. 中国科学院上海技术物理研究所,上海 200083;
3. 中国科学院大学,北京 100049;
4. 中国科学院红外探测与成像技术重点实验室,上海 200083)

摘要:基于深度学习的目标检测算法取得了很大成功,显著超越了传统算法,在很多场景下甚至可以和人类相媲美。不同于可见光相机,红外相机可以在黑暗环境下识别物体,可以用于安防和无人驾驶等领域。本文提出了面向嵌入式设备的轻量级目标检测算法,并采用赛灵思的Ultrascale+MPSoC ZU3EG FPGA加速并部署该算法。加速器运行在350 MHz的时钟频率下,吞吐量达到了551 FPS,功耗仅有8.4 W。在准确率方面,该算法在FLIR数据集下IoU指标达到了73.6%。在性能方面,相比于之前相同逻辑资源下性能最好的硬件加速器Ultranet,该加速器设计将吞吐量提高了2.59倍,功耗降低了2.04倍,降低至原来的49.02%。

关 键 词:红外图像处理;实时嵌入式系统;可编程逻辑器件;卷积神经网络

中图分类号:TN47 **文献标识码**:A

Introduction

Infrared systems have the unique advantage of resolving objects in dark environment or bad weather. Nowosielski *et al.* [1] took advantage of this feature and developed a system to expand human vision, which was

Received date: 2022- 01- 13, **revised date**: 2022- 07- 04 **收稿日期**:2022- 01- 13,**修回日期**:2022- 07- 04

Foundation items:Supported by the National Pre-Research Foundation of China during the "14th Five-Year Plan" (514010405-207)

Biography:CHEN Shaoyi(1995-), male, Quanzhou, master. Research area involves edge devices, deep learning hardware deployment, hardware and software co-design. E-mail: chenshy2@shanghaitech. edu. cn

[*] **Corresponding author**:E-mail: lizheng_sitp@163. com

used to detect pedestrians in driving at night to improve vehicle safety. Mushahar *et al.* [2] deployed an infrared system at the entrance of a public place to take contactless temperature measurements while detecting pedestrians and prohibiting people with excessive body temperature from entering the place.

Although the infrared image has many advantages, it lacks contrast and edge information compared with the visible image. Akula *et al.* [3] proposed WignerMSER to solve this problem, which is a new detector based on the local feature of infrared image, and is used to enhance the effect of target detection. Traditional image processing methods are often used in resource-constrained embedded systems because of low computational cost. In order to enhance the detection effect, a two-stage image processing method is generally adopted. Wu *et al.* [4] extracted the candidate box of pedestrians by using an adaptive threshold and vertical edge operator. Then, they carried out the bounding box in real-time of far-infrared pedestrians using the morphological method. Piniarski *et al.* [5] first preprocessed the image with two global thresholds to expand the region of interest and then performed pedestrian segmentation. Ragb *et al.* [6] also adopted a two-stage algorithm. Gradient information and texture information were used to obtain local information of the image. Then, the superpixel algorithm was used to find the detailed region without background information.

Deep learning shows more robust performance than traditional algorithms in the field of target detection and is widely used in the field of infrared target detection. Li *et al.* [7] proposed YOLO-FIRI based on YOLOv5 to solve the target detection problem of long distances, weak energy, and low resolution in infrared images. Yun *et al.* [8] combined neural network YOLO and Long Short-Term Memory (LSTM) to detect the problem of occluded infrared targets. Narayanan *et al.* [9] first used the YOLO network to extract features and then combined it with the support vector machine (SVM) classifier to classify (what).

Neural network algorithm has achieved good results in target detection, but there is huge challenge in embedded deployment. To deploy neural network algorithms on embedded devices usually has many limitations, including real-time, limited computing units, and restricted on-chip memory. Huang *et al.* [10] implemented the YOLO-tiny neural network accelerator, quantifying the parameters of the network to 2-bit for deployment of low-cost development boards. However, the computational performance of their accelerator only reaches 90. 6 GOP/s on the PYNQ-Z2 development board, which is difficult to be applied to practical scenarios. In order to deploy the neural network VGG16+SSD on the PYNQ-Z1 development board with fewer resources, Kang [11] reduced the weight by 87. 5% through accelerator-aware pruning. However, it is a challenge to achieve such a high pruning rate on all networks. In recent years, systolic array has been widely used in the accelerator design because of its high throughput. The potential for designing a much more efficient data path still remains to be explored. The performance density of the YOLO-tiny accelerator accelerated by Li[12] based on systolic arrays is only 0. 165 GOPS/DSP.

FPGA can generate the corresponding structure accelerator according to the low precision weight. This feature gives full play to all the advantages of the design. There have been many studies on deep learning accelerators based on FPGA. Lee et al. [13] proposed a structure of two multiply-accumulate (MAC) operations on one DSP. In this design, a subtraction of the left-shifted multiplier follows a double MAC processing unit. This operation is usually done by look-up tables (LUTs), where comes the bottleneck of the system. Fu et al. explored how to optimize operations in deep learning on DSP slices: multiplication and addition of integers with 8-bit width. However, their design only brought about a 1. 75× performance improvement compared ideally to a 2× improvement due to the bottleneck of DSP bit width. The existing accelerators could not thoroughly combine the characteristics of FPGA architecture from the studies above.

In order to solve this problem, we first put forward a lightweight neural network algorithm for infrared target detection. Then, we use high-level synthesis (HLS) to implement the convolutional neural network accelerator and deploy this algorithm on the Ultra96v2 development board. Finally, according to the characteristics of the FPGA structure, we realize the 2× MAC operation on a DSP. Experimental results show that when the input image resolution is 640*360 and the accelerator operating frequency is 350 MHz, the throughput of the accelerator reaches 551 FPS with 8. 4 W of power consumption. All in all, this paper designed a streaming-based accelerator to effectively deploy embedded infrared target detection algorithm.

The main contributions of this paper are summarized as follows:

- This paper implemented a pipeline style infrared target detection accelerator with high throughput using high level synthesis.
- This paper realized 2× MAC in a single DSP on FPGA.
- This paper significantly reduced the power of accelerator by using software and hardware co-optimization.

The remaining part of the paper proceeds as follows: The next section introduces the target network of acceleration briefly. Section II gives a basic introduction to the concept of high-level synthesis for hardware design. Section III describes our approach to hardware design in detail and the optimization method of the accelerator is delivered. Section IV is the experimental results and analysis. Section V summarizes our work.

1 Proposed object detection on thermal images

Although convolutional neural network algorithm is widely used in the field of target detection and has achieved good results, however, to pursue accuracy, current neural network models always comes with a large quantity of layers and parameters. These networks become increasingly unsuitable for the deployment of edge devices. Lightweight network, which aims to reduce the number and complexity of model parameters while maintaining model accuracy, has gradually become the focused research in computer vision.

As shown in Fig. 1, we set the backbone network of the accelerator to SkyNet[14]. SkyNet is a hardware-friendly and lightweight neural network for target detection and is the 2019 DAC-SDC champion model. In this model, the depthwise convolution[15] is used to replace the traditional convolution, which significantly reduces the computation and parameter of the model. In order to detect small targets, the bypass provides more low-level and high-resolution features to improve the target detection effect. SkyNet also uses YOLO′s detection head and two anchors to bound box regression.

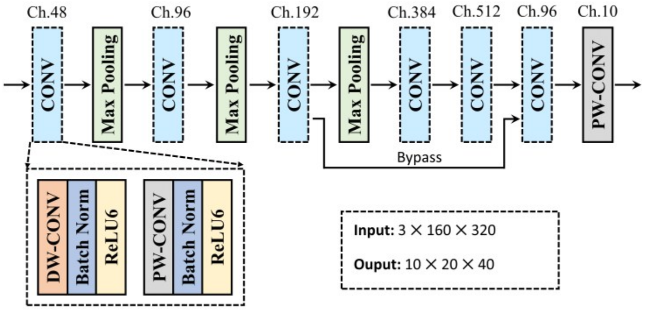

Fig. 1 The network structure of infrared target detection algorithm based on deep learning
图1 基于深度学习的红外目标检测算法网络结构

Network designed in top-down flow tends to have more layers and parameters. More parameters do not necessarily lead to better performance in a particular dataset. Table 1 compares IoU precision and parameter size between several classical networks and SkyNet in the DAC-SDC dataset. It can be found that the performance of network object detection with large parameters such as ResNet and VGG lags behind SkyNet in the DAC-SDC dataset. This situation shows that neural network is not just a simple algorithm that depends on large parameters, and it is possible to achieve object detection efficiently in the embedded device.

Table 1 SkyNet parameters and performance comparison with the classical network on DAC-SDC dataset
表1 SkyNet和经典网络参数量和性能在DAC-SDC数据集上的比较

Net name	ResNet-18	ResNet-34	ResNet-50	VGG-16	SkyNet
Parameter	11.18 M	21.28 M	23.51 M	14.71 M	0.44 M
IoU	0.61	0.26	0.32	0.25	0.73

SkyNet adopts the same detection method as YOLO for object detection. Firstly, the images of the input model were divided into 20×40 grids, and each grid predicted objects centered on the grid according to the anchor. In order to reduce calculation amount, this paper set the number of grids to two. Anchor with a fixed size is generally adopted in Fast-RCNN, but the method may not suit all objects with different sizes. In order to improve the training accuracy, a clustering algorithm is used to select anchors according to the dataset.

To simplify the hardware design, we used ReLU instead of ReLU6. The FLIR dataset is used for training, which contains 14,452 infrared images, including people, bicycles, cars, etc. and annotated with MSCOCO dataset format. We train 100 epochs on the training set with batch size 30. The initial learning rate is 1e-2, and the IoU reaches 73.6%. We quantify the network weight to 5 bits and the feature map to 8 bits, and the final IoU is 72.3%. The object detection results on the infrared dataset are shown in Fig 2. SkyNet has a good performance on infrared datasets. It can meet the requirement of embedded device target detection.

2 HLS preliminaries

Before we present our hardware design, we first review some basic concepts of high-level synthesis (HLS) design. HLS simplifies the development process of traditional hardware and uses C/C++ language to achieve the hardware design and development completed by traditional RTL.

As shown in Fig. 3, we use initial interval and latency to describe the performance of a hardware module. Latency is defined as the number of clock cycles required for the function to compute all output values. The smaller the number of clocks cycles, the better the hardware performs. To achieve the goal of reaching a smaller latency, more resources, sometimes intolerable resources are consumed. Initial interval (II) is defined as the number of clock cycles before the function can accept new input data. The little II is, the higher the throughput of the hardware is, the more circuit results can be obtained at the same time. However, this requires designing data path skillfully to achieve a pipelined circuit structure.

Take the neural network accelerator as an example. Without parallel design, each network layer can only run sequentially. In the whole design, latency and initial interval are equal. Each forward propagation can only wait to complete the last calculation before the module can start the consecutive calculation. In the process of waiting for the end of other stages, multiple computing units are idle, resulting in declining of performance degradation and computational efficiency.

When the pipeline is used for optimization, the circuit takes less time to receive new data and completes more tasks per clock cycle. Each stage of the pipeline is relatively independent. As long as there is uncalculated data in each stage, the circuit will continue to work. Balanced pipeline significantly improves the efficiency of computation so that the processing speed is greatly improved when dealing with continuous tasks.

The deep learning accelerator divides the pipeline by layers. Due to the massive difference in the workload

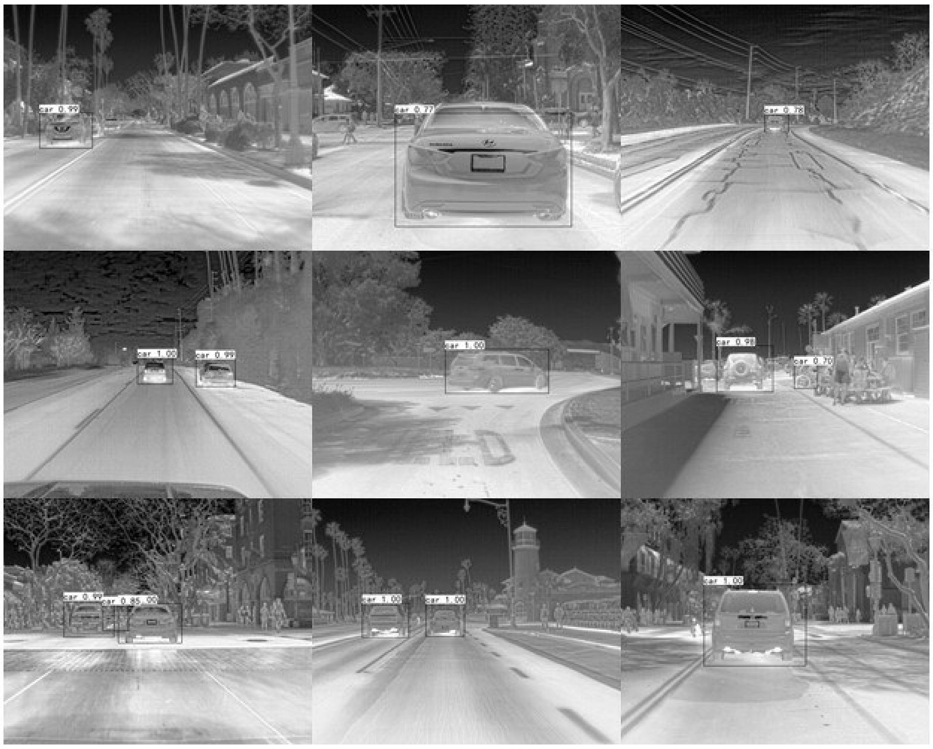

Fig. 2　SkyNet object detection result on FLIR dataset
图2　SkyNet在FLIR数据集上的检测结果

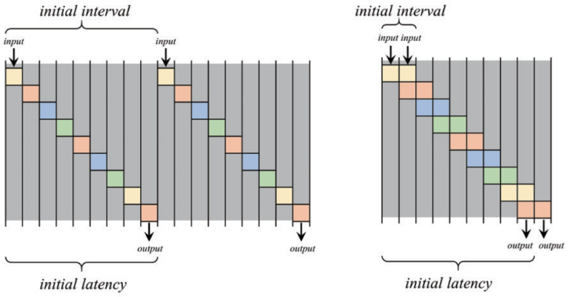

Fig. 3　Concepts of initial interval and latency
图3　初始间隔和延迟的定义

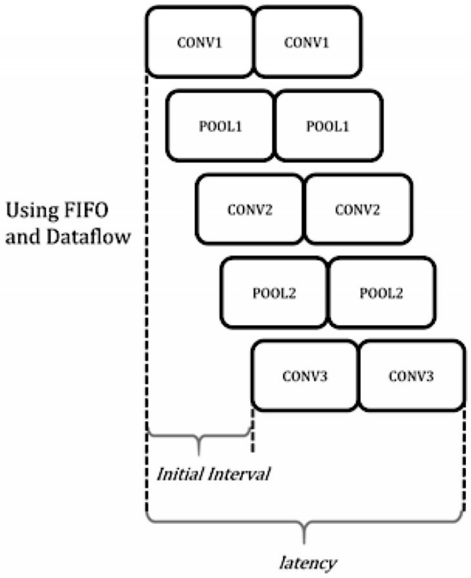

Fig. 4　The accelerator design for balancing all stages of pipeline
图4　各级流水线平衡的加速器设计

between different layers, there is a vast distinction of calculation amount in different stages of simple pipeline implementation. In order to solve this problem, we can place corresponding computing resources according to the amount of calculation in each layer so that the clock cycles of each pipeline stage are similar. As shown in Fig. 4, a balanced pipeline design makes the calculation efficiency of the circuit higher and reduces the idle time of the computing unit.

3　Hardware implementation

3.1　Overview of accelerator architecture

The accelerator is divided into two parts. CPU is in charge of reading image files, and FPGA is for realizing the deep learning accelerator with high parallelism. The data is exchanged between CPU and FPGA through the DMA module. The deep learning accelerator adopts pipeline structure, which is mainly composed of three parts: image pre-processing, convolution and max-pooling module. Divide the pipeline into stages with each convolution or max-pooling operation. Each convolutional or max-pooling layer is a computing unit, using FIFOs as interconnection. The neural network weights are stored in the on-chip block RAMs, and all the computing units work in parallel simultaneously to maximize resource uti-

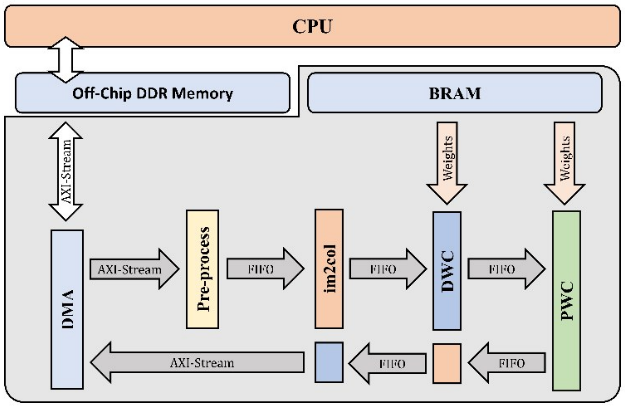

Fig. 5 FPGA inference accelerator architecture
图5 FPGA加速器架构

lization.

3.2 Balanced pipeline

The slowest stage determines the throughput of pipeline-style circuits. For pipeline-style circuits, the unbalanced pipeline will lead to the bubbles in the pipeline, leading to failure of computing units working at full speed. In other words, an unbalanced pipeline stage between different compute units indirectly leads to the waste of resources on the chip. The computation amount of different neural network layers is significantly different, making it very important to balance.

After dividing each convolution or max-pooling operation into a pipeline stage, we can approximate the complexity of hardware according to the number of input and output feature map channels, and the size of the convolution kernel.

The feature map is the intermediate result of operators at the convolution layer or max-pooling layer. We use W to represent the width of the feature map, H to represent the height of the feature map, C_{in} to represent the number of input channels, C_{out} to represent the number of output channels of the feature maps, and K to represent the size of the convolution kernel. When the stride is equal to 1, the computation amount of the depthwise convolution operation is

$$T_{DWC} = H \times W \times K \times K \times C_{in} \quad . \quad (1)$$

The computation amount of the pointwise convolution operation is

$$T_{PWC} = H \times W \times C_{in} \times C_{out} \quad . \quad (2)$$

We use the number of multiply-accumulate operations (MAC) to represent the computation of the convolution layer in the deep learning algorithm. $\#MAC_n$ is used to indicate the amount of computation at the nth layer. The parallelism is used to measure the multiply-accumulate operations that each clock can complete, and the parallelism PF_n represents the amount of computation at the nth layer. Throughput is used to evaluate the number of operator operations that can be done per second. T_n is used to represent the throughput of the nth layer. Assuming that the clock frequency running on the hardware is f_{clk}, T_n can be expressed as

$$T_n = \frac{PF_N}{\#MAC_n} \times f_{clk} \quad . \quad (3)$$

It is an upper bound on the reachable throughput of each accelerator operator. PF_{accel} is used to represent the parallelism of the accelerator, and is defined as:

$$PF_{accel} = \max\left(PF_i\right), i \in 1,2,\cdots,N \quad . \quad (4)$$

Assuming that the accelerator has only one clock domain and using the condition of pipeline balance $T_1 = T_2 = \cdots = T_N$, the theoretical parallelism of the nth layer can be obtained by the following equation:

$$PF_n = \frac{\#MAC_n}{\max\left(\#MAC_i\right)} \times PF_{accel}, i \in 1,2,\cdots,N \, . \quad (5)$$

When the result is a fraction, it is necessary to round up to obtain the parallelism of integers, thus simplifying the design of accelerators. Rounding up introduces only minor pipeline mismatches, and the loss of resource utilization is negligible, having no impact on performance.

3.3 Accelerator design based on high level synthesis

Datapath needs to be carefully designed for the hardware to deploy deep learning algorithms on resource-limited embedded devices effectively. In order to make full use of on-chip resources, all weights are stored on the chip, and due to the bandwidth of on-chip memory, the time of weight accessing can be ignored. In order to minimize data caching buffer size between neural network layers, our design reuses feature map as much as possible. All intermediate data is channel continuous as it passes. We introduce the realization of each operator respectively below and assuming that the size of the output feature map of the previous layer is $H \times W \times N\_ICH$.

3.3.1 Pointwise convolution datapath design

Fig. 6 shows the design of pointwise convolution data path. We divide the feature map into many sliding cubes with sizes of $1 \times 1 \times N\_IN$. Each time computing unit multiplies and accumulates with one cube in the feature map and N_IN weights of N_OUT channels. It is equivalent to having the parallelism of the accelerator set to $N\_IN \times N\_OUT$. PW unit takes N_ICH/N_IN clock cycles to compute the results of N_OUT consecutive channels and passes them to the next layer. PW unit takes

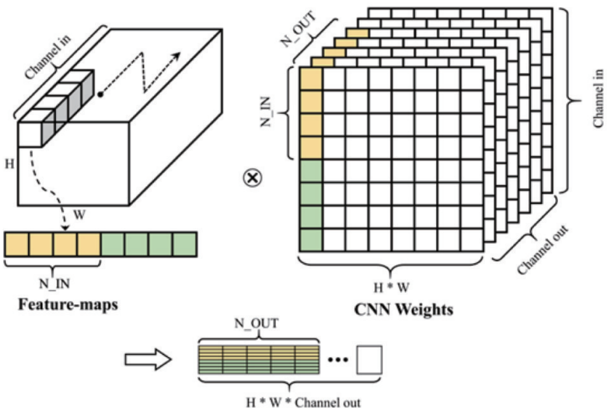

Fig. 6 Datapath of pointwise convolution
图6 点卷积的数据通路

N_OCH × *H* × *W* / *N_OUT* computations to get the result of PW convolution. The pseudocode for PW convolution is as follows：

Algorithm 1 Pseudocode for Pointwise Convolution Layer

```
Input: in<N_IN × BIT_IN>: feature map input
weight< N_OUT × N_IN × BIT_WT> [N_OCH / N_OUT][N_ICH / N_IN] : weight of neural network
N_IN: number of input parallel factor
N_OUT: number of output parallel factor
N_ICH: number of input channel
N_OCH: number of output channel
BIT_IN: bitwidth of input
BIT_WT: bitwidth of weight
BIT_OUT: bitwidth of output
#pragma HLS DATAFLOW
for fo = 0; fo<N_OCH / N_OUT ; ++fo do
    for fi = 0; fi<N_ICH / N_IN ; ++fi do
        #pragma HLS PIPELINE II=1
        for i = 0; i<N_IN ; ++i do
            #pragma HLS UNROLL
            for o = 0; o<N_OUT ; ++o do
                out<SLICE(BIT_OUT, o)> += in<SLICE(BIT_IN, i)> *
                weight<SLICE(BIT_WT, N_IN * o + i)>[fo][fi];
            end for
        end for
    end for
end for
Output: out<N_OUT * BIT_OUT> : feature map output
```

3. 3. 2　Depthwise convolution datapath design

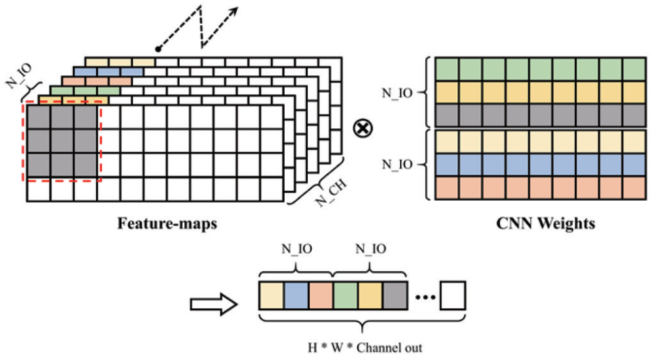

Fig. 7　Datapath of depthwise convolution
图 7　深度卷积的数据通路

As is shown in Fig. 7 and Fig. 8, the calculation of depthwise is divided into two steps. First, line buffer and window buffer are used to reduce the bottleneck of memory access, and the feature map within the range of 3×3 convolution kernel size on *N_IO* channels is converted into data flow with *N_IO*×*BIT_WIDTH* and depth of 9.

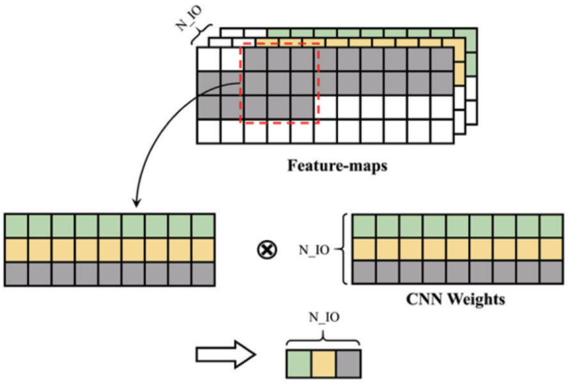

Fig. 8　Using line buffer to optimize datapath
图 8　使用 line buffer 优化深度卷积的数据通路

The DW unit gets one-weight data from memory with data width of *N_IO*×*WEIGHT_WIDTH* and depth of 9 and then multiplies the feature map with the pre-ordered weights in the DW unit to obtain the N_IO channels result. The result of DW convolution can be obtained by *N_OCH* × *H* × *W* / *N_IO* times computation. The pseudocode of DW Conv is as follows：

Algorithm 2 Pseudocode for Depthwise Convolution Layer

```
Input: in<N_IO×BIT_IN> : feature map input
weight <N_IO×BIT_WT>[N_CH / N_IO][9] : weight of neural network
N_IO : number of input parallel factor
N_CH : number of input channel
BIT_IN : bitwidth of input
BIT_WT : bitwidth of weight
BIT_OUT : bitwidth of output
#pragma HLS DATAFLOW
for f = 0; f < N_CH / N_IO; ++f do
    for k = 0; k<9; ++k do
        #pragma HLS PIPELINE II=1
        wt_buf = weight[f][k]
        for i = 0; i<N_IO; ++i do
            #pragma HLS UNROLL
            for o = 0; o<N_OUT ; ++o do
                out<SLICE(BIT_OUT, i)> += in<SLICE(BIT_IN, i)> *
                wt_buf <SLICE(BIT_WT, i)>;
            end for
        end for
    end for
end for
Output: out<N_IO * BIT_OUT> : feature map output
```

3. 3. 3　Maxpool datapath design

The design of the pipeline structure of max-pooling is shown in the Fig. 9. In this design, the stride of max-pooling is 2, and the operation is performed in two main steps, pool1D and pool2D. Pool1D is in charge of maximum pooling in the horizontal direction, comparing every

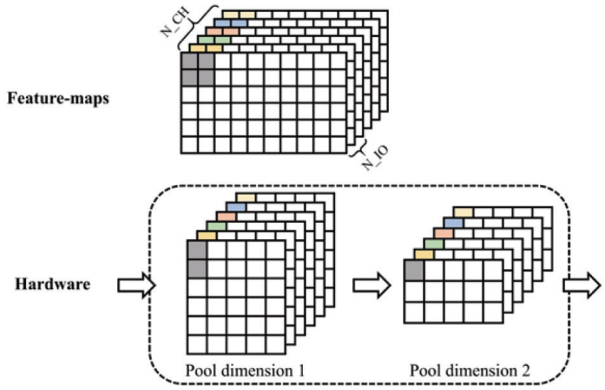

Fig. 9 Datapath of maxpool
图9 最大池化的数据通路

two received data (including N_CH channels) and outputting one data. Pool2D performs the maximum of vertical pooling. Assuming that the number of rows in the feature map starts at 0, Pool2D first caches data from even rows and outputs the final result after receiving one of the odd rows. Max-pooling module works in parallel with multiple channels, and the amount of parallelism depends on the number of PW's output channels.

3.4 Double MACs

Table 2 shows that convolution requires a lot of multiplication and addition operations. MAC relies on the limited DSP core of the FPGA, which provides better performance and lower power consumption. Although the FPGA can use LUTs to realize multipliers, it also causes timing issue. In order to improve the performance of the system on the limited computing resources and embedded devices, we implement a high-performance DSP reuse technology.

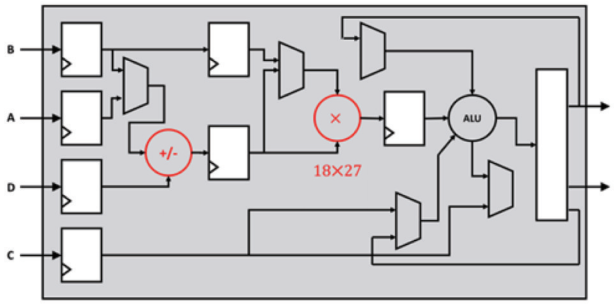

Fig. 10 DSP48E2 slice architecture
图10 DSP48E2的硬件架构

The DSP slice DSP48E2 in Xilinx FPGA is capable of completing a 27×18 bit width multiplication operation and 48-bit accumulation operation. Our network's feature map and weights are quantized to low bits (8 and 5 bits), making it possible for a DSP slice to perform two low-width MACs in a single stage. We can use a high-width multiplication to take place of two low-width multiplications and split the result into two numbers.

Fig. 10 shows the internal structure of Xilinx's DSP48E2 slice. Two different weights are input from port A and port D into the pre-adder which concatenates the two weights. The feature map is input to the multiplier from port B, and the result of the feature map and the spliced data after the multiplication operation is sent to the accumulator. If there is data to be accumulated, the data is input to the accumulator through port C.

DSP units used to complete vector MACs and the calculation result of each unit is transmitted to the port C of the next DSP. The vector I_0 and two vectors W_0 and W_1 align through delay and finally input to the PE. W_1 needs sign extension and left shifting. The output Y can be expressed as

$$Y = (W_1 \ll 14 + W_0) \times I_0 \quad . \quad (6)$$

The addition and multiplication are implemented by the pre-adder and multiplier in the DSP block, respectively. The cascaded adder implements the accumulation. Reorganize (6) to get

$$Y = W_1 \times I_0 \ll 14 + W_0 \times I_0 = O_1 \ll 14 + O_0, \quad (7)$$

O_0 and O_1 are the dots product results. Assuming that O_0 and O_1 have a bit width of 13, and O_0 is equal to the low part of Y , while O_1 will be affected by the sign bit of O_0 and needs to be corrected.

$$O_0 = Y[12:0] \quad , \quad (8)$$

$$O_1 = Y[26:14] + Y[13] \quad . \quad (9)$$

Only a 13 bit carry chain is needed to implement the result correction. Our design requires a small amount of extra logic while doubling the DSP utilization.

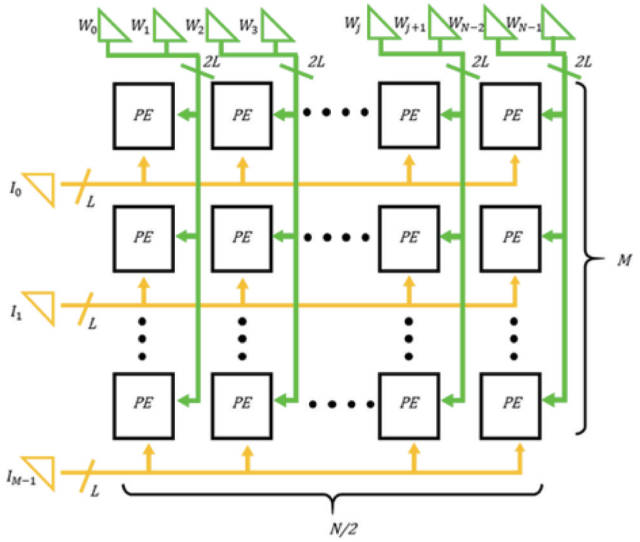

Fig. 11 Datapath of process element array
图11 处理单元阵列的数据通路

Fig. 11 shows how the PE array in pointwise convolution works. The yellow line represents the feature map passed in from the previous layer, and the green line is the input weight of parameter. With 2× MACs, both weights enter a PE simultaneously.

3.5 System optimization

Fig. 12 is the optimal method for the whole system test. The test picture data is saved on the SD card. In order to optimize the bottleneck of image reading, multi-threading is used to reduce the delay of image reading on

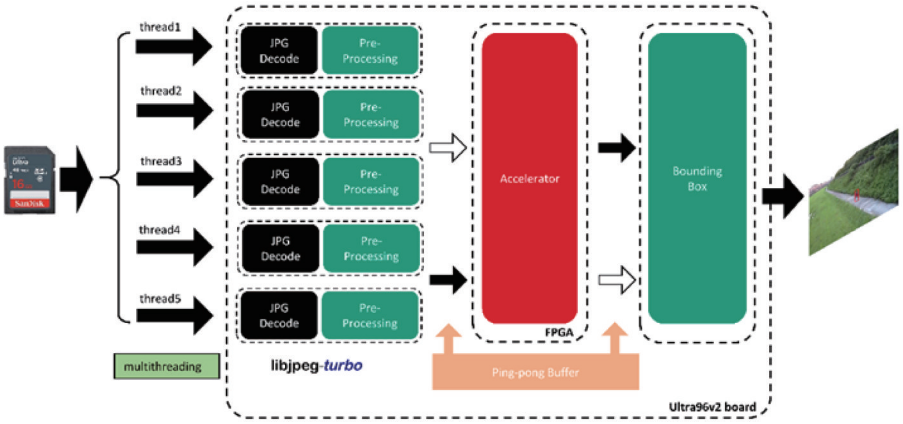

Fig. 12 System optimization
图12 系统优化

the Linux operating system, and the libjpeg-turbo library is used to speed up image decoding. Ping-pong buffer reduces the system bottleneck when the accelerator on FPGA communicates with the programmable system.

4 Experiment and analysis

4.1 Experimental setup

We deploy the deep learning accelerator on the Ultra96v2 evaluation board based on the SkyNet network. Ultra96v2 evaluation board is equipped with a Xilinx Zynq Ultrascale+MPSoC ZU3EG hybrid chip, mainly composed of CPU and FPGA. The logical part of the chip is 16 nm FPGA, which has 7.6 Mb BRAMs, 360 DSPs, and 70.6 K LUTs. The CPU part contains quad ARM Cortex-A53 CPU. Use the IP driver provided by the PYNQ framework to control the accelerator. As is shown in Table 2, *K* represents the size of the kernel, *C* represents the number of output channels at this layer, *FM* represents the size of the feature map, *#MAC* represents the amount of computation, *PF* represents the parallel factor. SkyNet consists of six depthwise convolutions and seven pointwise convolutions. The 10th convolution layer has the most extensive computation, and its parallelism is set to 256. The parallelism of other layers is set according to Eq. (5).

Part of the parallel factor is rounded up to simplify the design of the accelerator. The specific settings are shown in the Table 2. Thanks to the weights of the original neural network indicated by floating-point being quantized to 5 bits and the activation to 8 bits, and the weight size of the entire network is reduced to 277 KB, making it possible to store all the weight data on the chip. The intersection over union (IoU) of the original network can reach an accuracy of 73.6% on the same data set, while the quantized model is reduced by 1.3% to 72.3%. Our accelerator consumes 206.5 block RAMs (BRAM), 360 DSPs, 50518 look-up-tables (LUT), and 40488 flip-flops (FF). The accelerator has no timing issue, and the design can run at 350 MHz clock frequency.

Table 2 Skynet's parallelism factors of each layer
表2 SkyNet神经网络并行度设置

Layer	Type	K	C	FM	#MAC	PF
1	DW	3	3	160×320	1382400	3
2	PW	1	48	160×320	7372800	12
3	DW	3	48	80×160	5529600	12
4	PW	1	96	80×160	58982400	96
5	DW	3	96	40×80	2764800	6
6	PW	1	192	40×80	58982400	96
7	DW	3	192	20×40	2764800	3
8	PW	1	384	20×40	58982400	96
9	DW	3	384	20×40	2764800	6
10	PW	1	512	20×40	157286400	256
11	DW	3	1280	20×40	9216000	16
12	PW	1	96	20×40	98304000	160
13	PW	1	10	20×40	768000	2
Total					465100800	764

Table 3 Comparison with DAC-SDC accelerator design
表3 和DAC-SDC竞赛加速器设计对比

	iSmart	BJUT Runner	SkrSkr	Our work
Model	SkyNet	UltraNet	SkyNet	SkyNet
# of MACs	465M	272M	465M	465M
# of PFs	256	448	512	764
Frequency(MHz)	220	166	333	350
BRAMs	209	150.5	209	206.5
DSPs	329	360	329	360
LUTs	53809	44633	52875	50518
FFs	55833	58813	55278	40488
Precision(W,A)	11,9	4,4	6,8	5,8
IoU	73.1%	65.6%	73.1%	72.3%
Throughput(FPS)	25	213	52	551
Power(W)	13.5	6.66	6.7	8.4
Energy(mJ/img)	540	31	128	15.2

4. 2 Accelerator throughput analysis

The neural network SkyNet's forward propagation requires 465 MMAC operations. Our accelerator is designed to do 764 multiplications per clock cycle. When the accelerator runs at a 350 MHz clock frequency, the theoretical throughput should be 764×350 MHz = 267. 4 GMACs. The actual test results of the accelerator in Table 3 show a frame rate of 551 FPS and a throughput of 256. 2 GMACs. The computational efficiency of the accelerator is 256. 2 / 267. 4 = 95. 8%. Such high computational efficiency mainly benefits from the balanced pipeline structure and the ultra-high bandwidth of on-chip memory. Overall, our accelerator achieves a computational efficiency of 95. 8% at 350 MHz.

4. 3 Comparison with other works

Comparing our design with the design of the previous design automation conference-system design contest (DAC-SDC) winners in Table 3, we can find the superiority of the accelerator design in this paper. The experimental data are from the official website of DAC-SDC. BJUT Runner is the champion of DAC-SDC 2020. They also adopted the pipeline-style accelerator design, but the unreasonable pipeline-style setting of their accelerator made it unable to give full play to the advantages of pipeline-style. Our work is 2. 59× better in performance and 2. 04× better in energy efficiency than theirs. The computational efficiency was only 77. 8% with theoretical throughput of 448×166 MHz = 74. 4 GMACs/s and actual throughput of 57. 9 GMACs/s.

SkrSkr was the second place in DAC-SDC 2020. Their design used a convolution computing engine with very high parallelism in FPGA to calculate one convolution of a fixed size each time. The neural network layers were calculated sequentially, and the intermediate results and the weight of the neural network were passed repeatedly between the on-chip memory and DDR, resulting in performance degradation. Our design has a 10. 6× throughput and 8. 4× power consumption improvement over SkrSkr.

5 Conclusion

This paper presents a low-power convolutional neural network accelerator for infrared target detection on embedded devices. A design method to minimize hardware resource consumption is proposed for this hardware structure. The accelerator achieves 249. 7 GMACs throughput on the Xilinx ZU3EG device and achieves the best computational efficiency and energy efficiency compared to previous works. This low-powered deep learning FPGA accelerator has high practical value.

References

[1] Nowosielski A, Małecki K, Forczmański P, *et al.* Embedded Night-Vision System for Pedestrian Detection [J]. *IEEE Sensors Journal*, 2020, **20**(16): 9293-9304.

[2] Mushahar M F A, Zaini N. 2021 11th IEEE International Conference on Control System, *Computing and Engineering* (*ICCSCE*), 2021: 222-227.

[3] Akula A, Ghosh R, Kumar S, *et al.* WignerMSER: Pseudo-Wigner Distribution Enriched MSER Feature Detector for Object Recognition in Thermal Infrared Images [J]. *IEEE Sensors Journal*, 2019, **19** (11): 4221-4228.

[4] Wu D, Wang J, Liu W, *et al. 2017 First International Conference on Electronics Instrumentation & Information Systems* (*EIIS*), **2017**: 1-4.

[5] Piniarski K, Pawłowski P. 2017 Signal Processing: Algorithms, *Architectures*, Arrangements, and Applications (SPA), 2017: 160-165.

[6] Ragb H K, Aspiras T H, Asari V K. *2018 IEEE International Symposium on Technologies for Homeland Security* (*HST*), **2018**: 1-7.

[7] Li S, Li Y, Li Y, *et al.* YOLO-FIRI: Improved YOLOv5 for Infrared Image Object Detection [J]. *IEEE Access*, 2021, **9**: 141861-141875.

[8] Yun S, Kim S. 2019 19th International Conference on Control, *Automation and Systems* (*ICCAS*), 2019: 94-96.

[9] Narayanan A, Kumar R D, RoselinKiruba R, *et al.* 2021 Sixth International Conference on Wireless Communications, *Signal Processing and Networking* (*WiSPNET*), 2021: 431-434.

[10] Huang J, Yang J, Nui S, *et al. 2021 China Semiconductor Technology International Conference* (*CSTIC*), **2021**: 1-3.

[11] Kang H J. 2019 International Conference on Field-Programmable Technology (ICFPT), 2019: 419-422.

[12] Li Y, Lu S, Luo J, *et al. 2019 IEEE 4th International Conference on Signal and Image Processing* (*ICSIP*), **2019**: 335-339.

[13] Lee S, Kim D, Nguyen D, *et al.* Double MAC on a DSP: Boosting the Performance of Convolutional Neural Networks on FPGAs [J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019, **38**(5): 888-897.

[14] Zhang X, Lu H, Hao C, *et al.* SkyNet: a hardware-efficient method for object detection and tracking on embedded systems [J]. *Proceedings of Machine Learning and Systems*, 2020, **2**: 216-229.

[15] Howard A G, Zhu M, Chen B, *et al.* Mobilenets: Efficient convolutional neural networks for mobile vision applications [J]. *arXiv preprint arXiv*: 170404861, 2017.